

Memory-Centric Computing: Recent Advances in Processing-in-DRAM

Onur Mutlu, Ataberk Olgun, Geraldo F. Oliveira, Ismail E. Yuksel
ETH Zürich

Abstract—*Memory-centric computing* aims to enable computation capability in and near all places where data is generated and stored. As such, it can greatly reduce the large negative performance and energy impact of data access and data movement, by 1) fundamentally avoiding data movement, 2) reducing data access latency & energy, and 3) exploiting large parallelism of memory arrays. Many recent studies show that memory-centric computing can largely improve system performance & energy efficiency. Major industrial vendors and startup companies have recently introduced memory chips with sophisticated computation capabilities. Going forward, both hardware and software stack should be revisited and designed carefully to take advantage of memory-centric computing.

This work describes several major recent advances in memory-centric computing, specifically in *Processing-in-DRAM*, a paradigm where the operational characteristics of a DRAM chip are exploited and enhanced to perform computation on data stored in DRAM. Specifically, we describe 1) new techniques that slightly modify DRAM chips to enable both enhanced computation capability and easier programmability, 2) new experimental studies that demonstrate the functionally-complete bulk-bitwise computational capability of real commercial off-the-shelf DRAM chips, without any modifications to the DRAM chip or the interface, and 3) new DRAM designs that improve access granularity & efficiency, unleashing the true potential of Processing-in-DRAM.

I. MEMORY-CENTRIC COMPUTING

Data movement between computation units (e.g., CPUs, GPUs, ASICs) and main memory (e.g., DRAM) is a major *performance* and *energy bottleneck* in current processor-centric computing systems [1–28], and is expected to worsen due to the increasing data intensiveness of modern applications, e.g., machine learning [14, 29–38] and genomics [39–47]. To mitigate the overheads caused by data movement, various works propose Processing-in-Memory (PiM) architectures [13, 14, 24–26, 31, 40, 42, 43, 45, 48–201]. There are two main approaches to Processing-in-Memory (PiM) [8, 10, 11]: (i) Processing-near-Memory (PnM) [13, 14, 24–26, 42, 43, 45, 48–119, 189, 202, 203], where computation logic is added near the memory arrays (e.g., in a DRAM chip, next to each bank, or at the logic layer of a 3D-stacked memory [13, 14, 24–26, 31, 40, 82, 88, 204–209]); and (ii) Processing-using-Memory (PuM) [90, 120–188, 190, 194, 210], where computation is performed by exploiting the analog operational properties of the memory circuitry.

Both approaches, offering different tradeoffs, are important to exploit the full potential of PiM. PuM has two major advantages over PnM: 1) PuM *fundamentally* reduces data movement by performing computation *in situ*, while data movement still occurs between computation units and memory arrays in PnM; 2) PuM exploits the large internal bandwidth and parallelism available *inside* the memory arrays, while PnM is bottlenecked by the memory’s internal data buses. In contrast, PnM can enable a wider set of functions (including complete processors) to be more easily implemented and exploited near memory due to its use of conventional logic.

PiM (both PuM and PnM) can be implemented in (i.e., using or near) different memory technologies [8, 10, 11], including SRAM (e.g., [60, 68, 71, 132–134]), DRAM (e.g., [13, 14, 24–26, 42, 45, 48–59, 61–66, 69, 70, 72–79, 82–86, 88–120, 122–128, 131, 135–138, 140, 146–153, 157, 159, 211]), NAND flash (e.g., [43, 44, 143, 160–171, 212]), or emerging (e.g., [80, 121, 129, 130, 139, 141, 144, 145, 155, 156, 158, 194]). We focus on DRAM [213] due to its dominance as the main memory technology and very large capacity that can house many data-intensive workloads at reasonably low access latency.

II. PROCESSING-IN-DRAM

Many works demonstrate Processing-near-DRAM (PnD) and Processing-using-DRAM (PuD).

PnD architectures are designed to accelerate important applications by adding specialized or general-purpose compute units next to DRAM banks or arrays, either inside the DRAM chip or inside the logic layer of 3D-stacked DRAM architectures. Prominent examples include acceleration of graph analytics [24, 25, 72, 90], machine learning [14, 29–37, 197, 214], mobile workloads [13], genome analytics [42, 45, 46, 199], databases [73, 76, 77, 82, 86, 88, 89], climate modeling [54, 201], time series analysis [91], security functions [81, 196, 215, 216], data manipulation [13, 85, 100], and GPU workloads [26, 84].

PuD architectures use DRAM to perform primitive operations (e.g., data copy, initialization, bitwise operations), on top of which different applications and software stacks can be built. We highlight a few major works we build on. RowClone [125] demonstrates that data copy and initialization can be accelerated within a DRAM subarray by performing two consecutive row activations, which lead to data in the first activated row to be copied to the second activated row via the sense amplifiers that are connected to both rows.¹ Ambit [122, 123, 138] demonstrates that i) concurrently activating three DRAM rows leads to the computation of the bitwise MAJority function (and thus the AND and OR functions) on the contents of the three rows due to the charge sharing principles that govern the operation of the shared bitlines and sense amplifiers (Fig. 1a), ii) bitwise NOT of a row can be performed through the sense amplifier, with modifications to DRAM circuitry (Fig. 1b). Ambit provides a DRAM chip architecture that can exploit such triple-row activation (TRA), NOT, and RowClone operations. SIMDRAm [185] shows that, via a new software/hardware cooperative framework (Fig. 2), *any* operation (e.g., multiplication, division, convolution) that can be expressed as a logic circuit consisting of AND, OR, NOT gates can be implemented and seamlessly programmed using the Ambit substrate.

Many operations envisioned by these PuD works can *already* be performed in *real unmodified* commercial off-the-shelf (COTS) DRAM chips, by violating manufacturer-recommended DRAM timing parameters. Recent works show

¹This work also demonstrates that DRAM operational principles can be used to copy data between different banks and subarrays, which later work [27, 159, 184] improves by adding further logic to facilitate.

that COTS DRAM chips can perform 1) data copy & initialization [131, 150] (as in RowClone [125]), 2) three-input bitwise MAJ and two-input AND & OR operations [131, 217, 218] (as in Ambit [122, 123, 126, 135, 136, 138]), and 3) true random number generation & physical unclonable functions [146–148].

III. CHALLENGES & OVERVIEW

To realize the full potential and benefits of Processing-in-DRAM (PiD), and more generally PiM, a number of important challenges need to be solved [8, 10, 11]. This work tackles several of these major challenges.

First, enabling widespread use of PiM on a wide variety of important workloads requires PiM systems to i) be easy to program and seamlessly compile workloads into [8, 10, 11] and ii) support a wide range of computation primitives and capabilities. Second, it is important to demonstrate the potential feasibility and capabilities of future PiM architectures, especially PiM ideas that exploit analog operational capabilities of memory chips, ideally on real hardware. Third, it is important to design the memory (e.g., DRAM) chip architectures to efficiently support processing capability in a programmable manner.

We highlight several recent works [219–223] that tackle these challenges for PiD systems: 1) MIMDRAM [219], for enabling easier programmability and enhanced computation capability, 2) new experimental studies using commercial off-the-shelf (COTS) DRAM chips [220, 221, 223], which demonstrate previously-unknown computational capabilities of real unmodified DRAM chips, and 3) Sectored DRAM [222], a new fine-grained DRAM design, that enables an efficient and easier-to-program DRAM substrate for PiM.

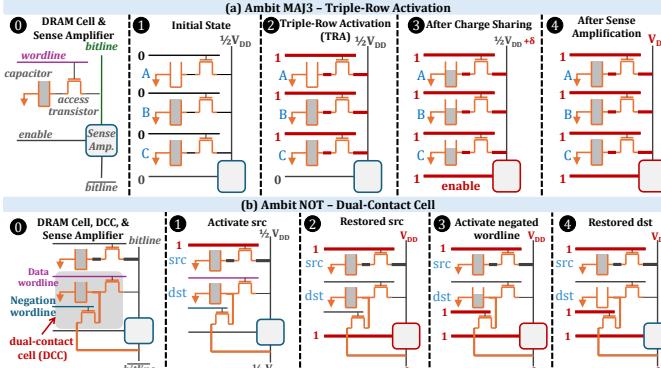


Fig. 1: An example of performing the MAjority-of-three operation (i.e., MAJ3 (A, B, C)) (a) and the NOT operation (i.e., dst=NOT (src)) in Ambit [122]. In (a), we focus on DRAM cell and sense amplifier operations (❶). Initially, cells A, B, C, and bitline have voltage levels of GND, VDD, VDD, and VDD/2, respectively (❷). We first perform a triple-row activation (TRA) to simultaneously activate cells A, B, and C (❸). When the wordlines of all three cells are raised simultaneously, charge sharing results in a positive deviation on the bitline because at least two of the cells are charged (❹). Therefore, after sense amplification, the sense amplifier drives the bitline to VDD, which then fully charges all three cells (❺). The final state of the bitline is, thus, the MAjority function of the charged state of the three cells A, B, and C. If one of the cells (say C) is set to GND (VDD), the final state would be the AND (OR) of the other two (A and B). To simplify the explanation, we assume no process variation and noise, but the Ambit paper and later works [122, 185] evaluate these effects. Ambit-NOT (b) introduces the dual-contact cell (DCC), which is a DRAM cell with two transistors. In a DCC, one transistor connects the cell capacitor to the bitline, i.e., data wordline, and the other transistor connects the cell capacitor to the bitline-bar, i.e., negation wordline (❻). Initially, src and dst cells each have a voltage level of VDD, and bitline and bitline-bar are precharged to VDD/2. To perform the NOT operation, we first activate the src cell (❻). The activation drives the bitline to the value corresponding to the src, VDD in this case, and the bitline-bar to the negated value, i.e., GND (❼). Second, Ambit activates the negation wordline. Doing so enables the transistor that connects the DCC to the bitline-bar. This results in the bitline-bar sharing its charge with the dst cell (❼). Since the bitline-bar is already at a stable voltage level of GND, it overwrites the value in the DCC capacitor with GND, thereby copying the negated value of the src cell into the dst cell (❼). The original Ambit paper (see Section 5 in [122]) proposes a DRAM subarray design that makes the implementation of triple-row activation low overhead by restricting TRA to an isolated set of DRAM rows that can be used for computation. It also describes circuit-level issues & system and programming support needed for Ambit and evaluates the hardware cost of modifications made to the DRAM chip and the memory controller. A later work [123] describes some outstanding issues in Ambit-like bulk-bitwise Processing-using-DRAM substrates.

IV. MIMDRAM

MIMDRAM is a hardware/software co-designed PuD system that introduces new mechanisms to allocate and control only the necessary resources for a given PuD operation. Major key ideas of MIMDRAM are 1) to leverage fine-grained DRAM (i.e., the ability to independently access smaller segments of a large DRAM row; see Section VI) for PuD computation, 2) enable near-subarray reduction computation logic across DRAM mats, and 3) provide compiler support to transparently map vector operations to DRAM mats and subarrays. MIMDRAM provides a multiple-instruction multiple-data (MIMD) execution model [224] in each DRAM subarray [21], enabling different DRAM mats within a subarray to execute different PuD instructions (where each PuD instruction specifies bit-serial SIMD execution within one or more DRAM mats). MIMDRAM eases programmability and compilation by reducing the minimum granularity of PuD operations to bit-widths commonly targeted by modern vectorizing compilers and enabling flexibility in computation granularity. Fig. 3 presents an overview of MIMDRAM. Fig. 4 presents an overview of the intra-mat interconnect in MIMDRAM. Fig. 5 shows an example PuD vector reduction operation using MIMDRAM.

We evaluate MIMDRAM using twelve real-world applications and 495 multi-programmed application mixes. When using 64 DRAM subarrays per bank and 16 banks for PuD computation in a DRAM chip, MIMDRAM provides 1) 13.2x/0.22x/173x the performance, 2) 0.0017x/0.00007x/0.004x the energy consumption, 3) 582.4x/13612x/272x the performance per Watt of the CPU [225]/GPU [226]/SIMDRAM [185] baseline (Fig. 6) and 4) when using a single DRAM subarray, 15.6x the SIMD utilization, i.e., SIMD efficiency (Fig. 7) of the prior state-of-the-art PuD framework, SIMDRAM.

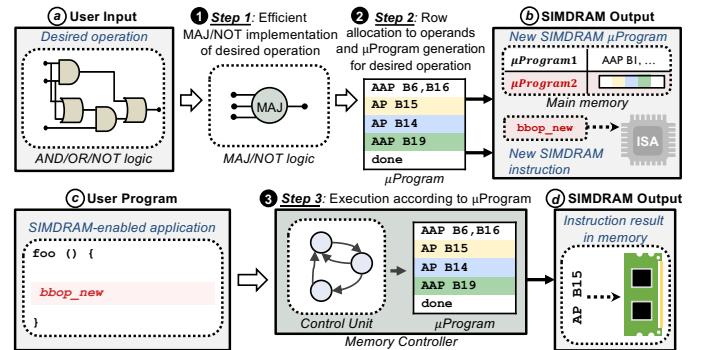


Fig. 2: Overview of the SIMDRAM framework [185]. SIMDRAM consists of three key steps to enable a user-specified desired operation in DRAM: 1) building an efficient MAJ/NOT-based representation of the desired operation, 2) mapping the operation input and output operands to DRAM rows and to the required DRAM commands that produce the desired operation, and 3) executing the operation. The first two steps give users the flexibility to implement and compute any desired operation in DRAM efficiently. The goal of the first step is to use logic optimization to minimize the number of DRAM row activations and, thus, the computation latency required to perform a specific operation. Accordingly, the first step (❶) takes as input the AND/OR/NOT-based implementation of the designed operation (labeled ❷ in the figure) and derives the operation's optimized MAJ/NOT-based implementation (i.e., the optimized majority inverter graph). The second step (❸) translates the optimized MAJ/NOT-based implementation into DRAM row activations, i.e., Ambit TRA [122] and RowClone [125] operations. This step includes 1) mapping the operands to the designated rows in DRAM and 2) defining the sequence of DRAM row activations required to perform the computation associated with the optimized MAJ/NOT implementation. SIMDRAM chooses the operand-to-row mapping and the sequence of DRAM row activations to minimize the number of DRAM row activations required for a specific operation. The output of the second step (❸) is stored as a microprogram (μ Program) in the memory controller, associated with the desired operation, $bbop_new$. The third step (❹) is to program the memory controller to issue the sequence of DRAM row activations to the appropriate rows in DRAM to perform the computation of the operation from start to end. When the user program (❺) encounters a SIMDRAM instruction (called $bbop_new$), the instruction is shipped to the memory controller, which invokes the associated μ Program and executes the operation as specified by the μ Program. To this end, SIMDRAM uses a control unit in the memory controller that transparently executes the sequence of DRAM row activations for each specific PuD operation executed by a user program. Once the μ Program is complete, the result of the operation (❻) is held in DRAM. Figure adapted from [185].

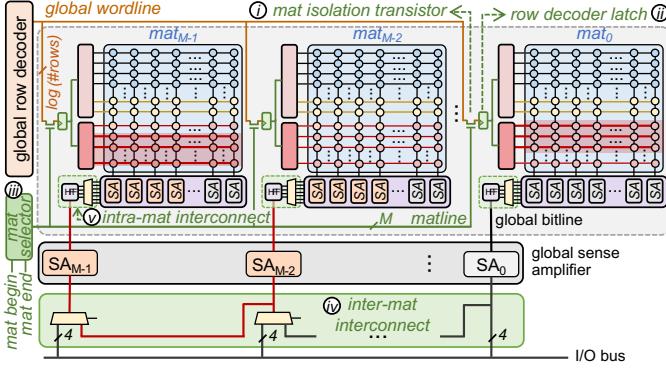


Fig. 3: Overview of the DRAM subarray and bank organization of MIMDRAM [219]. Green-colored boxes represent newly added or modified hardware components. To enable fine-grained PuD execution, MIMDRAM modifies Ambit's subarray and the DRAM bank with three new hardware structures: the *mat isolation transistor* (①), the *row decoder latch* (②), and the *mat selector* (③). At a high level, the *mat isolation transistor* allows for the independent access and operation of each DRAM mat within a subarray while the *row decoder latch* enables the execution of a PuD operation in a range of DRAM mats that the *mat selector* defines. MIMDRAM implements an *inter-mat interconnect* (④) to enable data movement across different mats by slightly modifying the connection between the I/O bus and the global sense amplifier. MIMDRAM adds a 2:1 multiplexer to each set of four 1-bit sense amplifiers in the global sense amplifier, selecting whether the data written to the sense amplifier set (SA_i) comes from the I/O bus or the neighboring sense amplifier set (SA_{i-1}). MIMDRAM enables data movement across columns within a DRAM mat through an *intra-mat interconnect* (⑤), which works by modifying the sequence of steps in the column access operation (hence without any hardware modification to the DRAM subarray structure). The intra-mat interconnect leverages the fact that 1) local bitlines in a mat already share an interconnection link via the helper flip-flops (HFFs) and 2) these HFFs can latch and amplify the local row buffer's data. Figure adapted from [219].

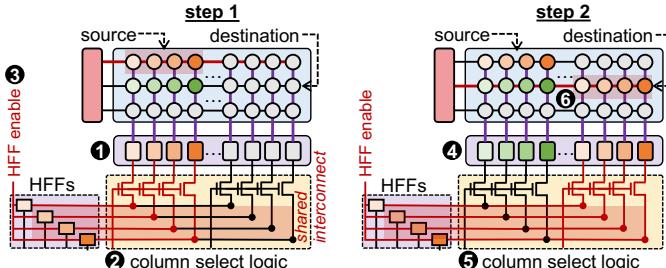


Fig. 4: Intra-mat data movement in MIMDRAM [219]. To enable data movement across columns within a DRAM mat, MIMDRAM implements an intra-mat interconnect (⑤ in Fig.3), which does not require any hardware modification. Instead, MIMDRAM modifies the sequence of steps DRAM executes during a column access to realize an intra-mat data movement operation. There are two key observations that enable the intra-mat interconnect. First, we observe that the local bitlines of a DRAM mat already share an interconnection path via the HFFs and column select logic (as this figure illustrates). Second, the HFFs in a DRAM mat can latch and amplify the local row buffer's data. To manage intra-mat data movement, MIMDRAM exposes a new DRAM command to the memory controller called LC-MOV (local I/O move). The LC-MOV command takes as input: (i) the logical mat range [mat_begin, mat_end] of the target row, (ii) the row and column addresses of the *source* DRAM row and column; and (iii) the row and column addresses of the *destination* DRAM row and column. With the intra-mat interconnect and new DRAM command, MIMDRAM can move four bits of data from a source row and column ($row_{src}, column_{src}$) to a destination row and column ($row_{dst}, column_{dst}$) in the same mat (mat_M). An LC-MOV command is transparently generated by MIMDRAM control unit based on the source and destination mat addresses in a *bbop_mov* instruction (which MIMDRAM compiler generates; see [219]): if the source and destination mats' addresses are the same, MIMDRAM control unit translates the data movement instruction into an LC-MOV command; otherwise, into a GB-MOV command (global I/O move); see [219] and Fig.5).

Once the memory controller receives an LC-MOV command, it performs two steps. In the first step, the memory controller performs an ACT-RD-PRE targeting $row_{src}, column_{src}$ in mat_M . The ACT loads row_{src} to mat_M 's local sense amplifier (①). The RD moves four bits from row_{src} , as indexed by $column_{src}$, into the mat's helper flip-flops (HFFs) by enabling the appropriate transistors in the column select logic (②). The HFFs are then enabled by transitioning the *HFF enable* signal from low to high. This allows the HFF to *latch* and *amplify* the selected four-bit data column from the local sense amplifier (③). The PRE closes row_{src} . Until here, the LC-MOV command operates exactly as a regular ACT-RD-PRE command sequence. However, differently from a regular ACT-RD-PRE, the LC-MOV command does not lower the *HFF enable* signal when the RD finishes. This allows the four-bit data from $column_{src}$ to reside in the mat's HFF. In the second step, the memory controller performs an ACT-WR-PRE targeting $row_{dst}, column_{dst}$ in mat_M . The ACT loads row_{dst} into the mat's local row buffer (④), and the WR asserts the column select logic to $column_{dst}$, creating a path between the HFF and the local row buffer (⑤). Since the *HFF enable* signal is kept high, the HFFs do *not* sense and latch the data from $column_{dst}$. Instead, the HFFs overwrite the data stored in the local sense amplifier with the previously four-bit data latched from $column_{src}$. The new data stored in the mat's local sense amplifier propagates through the local bitlines and is written to the destination DRAM cells (⑥). Figure adapted from [219].

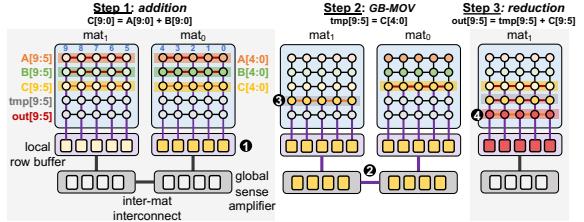


Fig. 5: An example of a PuD vector reduction, i.e., $out += A[i] + B[i]$, in MIMDRAM [219]. For illustration purposes, we assume that DRAM has only two mats, and the 10-bit data elements of the input arrays A and B are evenly distributed across the two DRAM mats. MIMDRAM executes a vector reduction in three steps. In the first step, MIMDRAM executes a PuD addition operation over the data in the two DRAM mats (①), storing the temporary output data C into the same mats where the computation takes place (i.e., $C = [C[9:5]_{mat_1}, C[4:0]_{mat_0}]$). In the second step, MIMDRAM issues a GB-MOV (global I/O move; a new DRAM command to perform inter-mat data movement) to move part of the temporary output $C[4:0]$ stored in mat_0 to a temporary row tmp in mat_1 ($tmp[9:5]_{mat_1} \leftarrow C[4:0]_{mat_0}$) via the inter-mat interconnect (②-③). MIMDRAM iteratively executes step 2 until all data elements of $C[4:0]$ are copied to mat_1 . In the third step, once the GB-MOV finishes, MIMDRAM executes the final addition operation, i.e., $tmp[9:5] + C[9:5]$, in mat_1 . The final output of the vector reduction operation is stored in the destination row out in mat_1 (④). Figure adapted from [219].

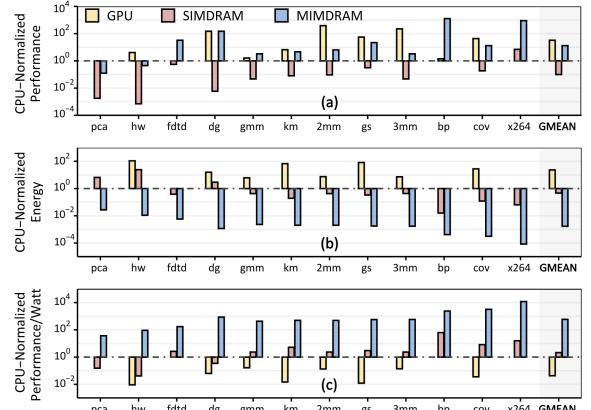


Fig. 6: CPU-normalized performance (a), energy (b), and energy efficiency (performance/Watt) (c) results for processor-centric (i.e., Intel Skylake CPU [225] and NVIDIA A100 GPU [226]) and memory-centric (i.e., SIMDRAM [185] and MIMDRAM [219]) architectures executing 12 real-world applications. We implement MIMDRAM and SIMDRAM using gem5 and evaluate their energy consumption using CACTI. We analyze 117 applications from SPEC 2017, SPEC 2006, Parboil, Phoenix, Polybench, Rodinia, and SPLASH-2 benchmark suites and identify 12 memory-bound, multi-threaded CPU applications where the most time-consuming loop can be auto-vectorized. These applications span various domains, including video compression, data mining, pattern recognition, medical imaging, and stencil computation. For this analysis, we allow SIMDRAM and MIMDRAM to fully leverage bank and subarray-level parallelism in a DRAM rank by allowing in-DRAM operations to happen simultaneously across all 16 banks and 64 subarrays per bank. MIMDRAM provides (1) $13.2 \times 0.22 \times 173 \times$ the performance, (2) $0.0017 \times 0.00007 \times 0.004 \times$ the energy consumption, and (3) $582.4 \times 1/13612 \times 272 \times$ the performance per Watt of the CPU/GPU/SIMDRAM baseline. In our analysis, we observe that MIMDRAM's end-to-end performance gains are limited by the throughput of the inter- and intra-mat interconnects, which are utilized during in-DRAM reduction operations. If we consider only MIMDRAM's arithmetic throughput (i.e., no reduction operations), we observe that MIMDRAM provides $272 \times$ and $11 \times$ the performance of the CPU and GPU baselines, respectively. We believe that combining PuD and PnD holistically, where auxiliary logic placed within the logic layer of 3D-stacked memories is used for high-throughput in-DRAM reduction and DRAM cells are used for high-throughput in-DRAM bulk arithmetic, would be beneficial to improve MIMDRAM's end-to-end performance. We conclude that MIMDRAM enables effective exploitation of DRAM bank-level and subarray-level parallelism for massively-parallel bulk bitwise execution.

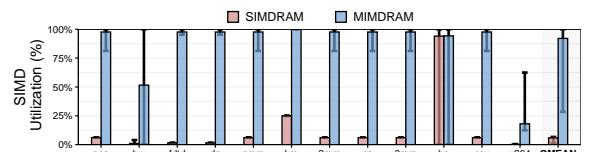


Fig. 7: SIMD utilization (i.e., the fraction of SIMD lanes executing a useful operation) of SIMDRAM and MIMDRAM for twelve real-world applications. Whiskers extend to the minimum and maximum observed date point values. On average, across all twelve real-world applications, MIMDRAM provides $15.6 \times$ the SIMD utilization of SIMDRAM. This is because MIMDRAM matches the available SIMD parallelism in an application with the underlying PuD resources (i.e., PuD SIMD lanes) by using only as many DRAM mats as the maximum vectorization factor of a given application's loop. In contrast, SIMDRAM always occupies all available PuD SIMD lanes (i.e., entire subarrays) for a given operation, resulting in low SIMD utilization for applications without a very-wide vectorization factor. We conclude that MIMDRAM greatly improves overall SIMD utilization for many applications.

V. CAPABILITIES OF REAL COTS DRAM CHIPS

A promising line of feasibility study of PuD systems is to understand the computation capabilities of existing DRAM chips via rigorous experimental testing. Multiple recent works [220, 221, 223] experimentally demonstrate various previously-unknown capabilities in unmodified DRAM chips. These capabilities arise from the operational principles of DRAM circuitry that are exercised by violating the manufacturer-recommended timing parameters [218, 227]. In particular, one can simultaneously activate *many* DRAM rows in state-of-the-art DRAM chips due to the hierarchical design of the row decoder circuitry [221, 223, 228–230]. Exploiting such simultaneous row activation, we [220, 221, 223] demonstrate that COTS DRAM chips are capable of 1) performing functionally-complete bulk-bitwise Boolean operations: NOT (Fig. 8), NAND, and NOR, 2) executing up to 16-input AND, NAND (Fig. 9), OR, and NOR operations, and 3) copying the contents of a DRAM row (concurrently) into up to 31 other DRAM rows (Fig. 10). We evaluate the robustness of these operations across data patterns, temperature, and voltage levels. Our results (Fig. 11) show that COTS DRAM chips can perform these operations at high success rates (>94%). These fascinating findings demonstrate the fundamental computation capability of DRAM, even when DRAM chips are *not* designed for this purpose, and provide a solid foundation for building new and robust PuD mechanisms into future DRAM chips and standards.

Simultaneous activation of multiple rows in DRAM can be used for generating true random numbers (TRNs) at high throughput (e.g., 3.44 Gb/s per DRAM channel [146]), widening the workloads supported by PiD systems (e.g., security-critical workloads) and enabling secure execution support for PuD systems that do *not* necessarily have dedicated TRN generation (TRNG) hardware (Fig. 12). Best prior TRNG using COTS DRAM chips generates TRNs by simultaneously activating four rows [146]. Our ongoing work experimentally studies the simultaneous activation of 2, 8, 16, and 32 rows in a subarray in COTS DRAM chips, showing that 8- and 16-row activation-based TRNG designs provide 1.25× and 1.06× higher throughput than the state-of-the-art (Fig. 13).

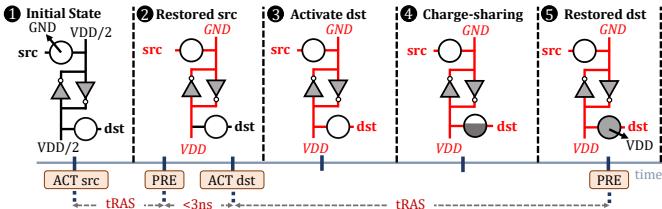


Fig. 8: Command sequence for performing the NOT operation ($dst = \text{NOT}(src)$) in COTS DRAM chips and the state of cells during each related step. The memory controller issues each command (shown in orange boxes below the time axis) at the corresponding tick mark on the time axis and asserted signals are highlighted in red. Cells initially have a voltage level of ground (GND), and the bitline (i.e., src 's bitline) and bitline-bar (i.e., dst 's bitline) initially have a voltage level of $VDD/2$ (1). NOT operation in COTS DRAM chips is performed in four key steps. First, we issue an ACT command to src , i.e., ACT src , and wait for the manufacturer-recommended t_{RAS} timing parameter to restore the charge of src . As a result, the bitline reaches the src voltage (GND), whereas the bitline-bar reaches the negated src voltage (VDD) (2). Second, we issue a PRE command and with violated manufacturer-recommended t_{RP} timing, e.g., <3ns, we issue another ACT command to activate dst , i.e., ACT dst . Issuing back-to-back PRE → ACT dst activates dst without deactivating src (3) and results in the bitline-bar sharing its charge with dst by driving the negated voltage value of src (VDD) into dst (4). Third, we wait for the manufacturer-recommended t_{RAS} timing parameter, which completely restores the charge of dst , and thus, the negated value of src is written to dst (5). Fourth, we send a PRE command to complete the process. Figure adapted from [220].

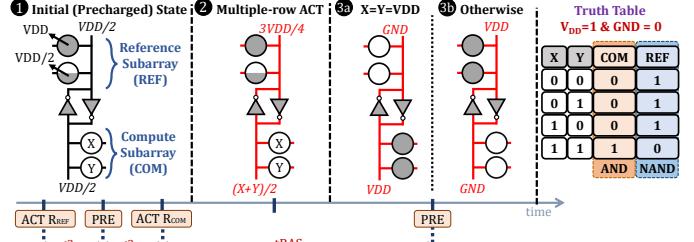


Fig. 9: Command sequence for performing the two-input AND and NAND operations (i.e., $\text{AND}(X, Y)$ and $\text{NAND}(X, Y)$) in COTS DRAM chips and the state of cells during each related step. The memory controller issues each command (shown in orange boxes below the time axis) at the corresponding tick mark and asserted signals are highlighted in red. In this figure, we have two neighboring subarrays: the *reference subarray* and the *compute subarray*, each containing two cells. To simplify the explanation, we assume that the bitline has no capacitance (i.e., after charge sharing, the bitline's voltage is the *mean* voltage value stored in DRAM cells that contribute to charge sharing). Assume that the ACT $R_{REF} \rightarrow \text{PRE} \rightarrow \text{ACT } R_{COM}$ command sequence with reduced timing simultaneously activates all four rows in these two subarrays where R_{REF} points to a row in the reference subarray and R_{COM} points to a row in the compute subarray. Initially, 1) we store VDD in one cell and $VDD/2$ in the other cell in the reference subarray, and 2) we store a voltage level of X in one cell and Y in the other cell in the compute subarray (1). To perform a two-input AND/NAND operation, we first issue one ACT $R_{REF} \rightarrow \text{PRE} \rightarrow \text{ACT } R_{COM}$ command sequence with violated timing parameters (1). Doing so activates four rows simultaneously and enables charge-sharing between their bitlines. At the end of charge-sharing, the reference subarray's bitline voltage (i.e., V_{REF}) becomes $3VDD/4$ (i.e., the mean of VDD and $VDD/2$), and the compute subarray's bitline voltage (i.e., V_{COM}) becomes $(X+Y)/2$ (2). The sense amplifier then kicks in and amplifies the voltage difference between V_{COM} and V_{REF} . If X and Y have VDD (i.e., $V_{COM}=VDD$), V_{COM} is higher than V_{REF} , which results in VDD in the compute subarray's activated cells and GND in the reference subarray's activated cells (3a). Otherwise, V_{COM} is lower than V_{REF} , which results in GND in the compute subarray's activated cells and VDD in the reference subarray's activated cells (3b). After waiting for t_{RAS} , we issue a PRE command to complete the two-input AND/NAND operation. As a result, activated cells in the compute subarray (COM) become the output of the $\text{AND}(X, Y)$ operation, and, at the same time, activated cells in the reference subarray (REF) become the output of the $\text{NAND}(X, Y)$ operation (as shown in the truth table where X and Y are the inputs and COM and REF are outputs). OR/NOR operations in COTS DRAM chips are similar in nature [220]. Figure adapted from [220].

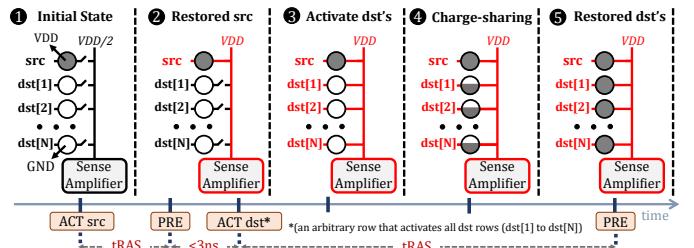


Fig. 10: Command sequence for performing the Multi-RowCopy operation (i.e., copying src row to N other dst rows simultaneously) in COTS DRAM chips and the state of cells during each related step. The memory controller issues each command (shown in orange boxes below the time axis) at the corresponding tick mark, and asserted signals are highlighted in red. Initially, src cell has VDD , dst cells (i.e., $dst[1]$ to $dst[N]$) have a voltage level of ground (GND) and bitline has a voltage level of $VDD/2$ (1). First, we issue an ACT command to src , i.e., ACT src , and wait for the manufacturer-recommended t_{RAS} timing parameter to restore the charge of src . As a result, the bitline reaches the src voltage (VDD) (2). Second, we issue a PRE command and with violated manufacturer-recommended t_{RP} timing, e.g., <3ns, we issue another ACT command to activate dst , i.e., ACT dst . The second ACT command interrupts the PRE command. By doing so, it 1) prevents the bitline from being precharged to $VDD/2$, 2) keeps src and the sense amplifier enabled, and 3) simultaneously activates dst cells. This results in the bitline sharing its charge with dst cells by driving the voltage value of src (VDD) into dst cells (4). Third, we wait for the manufacturer-recommended t_{RAS} timing parameter, which results in the sense amplifier overwriting all dst cells with src data (5). Fourth, we send a PRE command to complete the process. Multi-RowCopy operation can be used to accelerate not only data copy & initialization [125, 137] but also cold boot attack prevention as shown in [221].

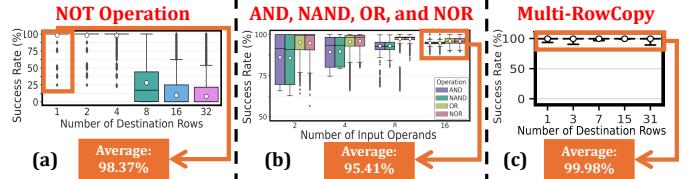


Fig. 11: Success rates of the NOT operation with varying numbers of destination rows (a) AND, NAND, OR, and NOR operations with varying numbers of input operands (b) the Multi-RowCopy operation with varying numbers of destination rows (c), as measured in 224, 224, and 120 COTS DRAM chips, respectively. On average, we observe a 98.37% success rate for the NOT operation with one destination row (a), 94.94%, 94.94%, 95.85%, and 95.87% for 16-input AND, NAND, OR, and NOR operations (b), and 99.98% for the Multi-Row Copy operation with 31 destination rows (c). We conclude that COTS DRAM chips can execute these operations with high reliability. More results and experimental methodology are in [220, 221].

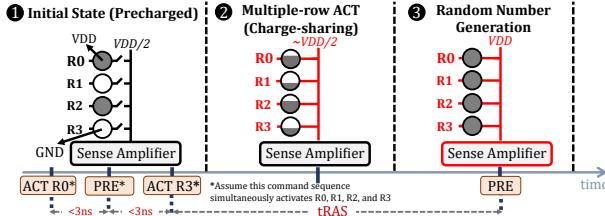


Fig. 12: Command sequence for true random number generation in COTS DRAM chips and the state of cells during each related step. The memory controller issues each command (shown in orange boxes below the time axis) at the corresponding tick mark and asserted signals are highlighted in red. Initially, two cells (R0 and R2) have a voltage level of V_{DD} , and the remaining two cells (R1 and R3) ground (GND), and bitline has a voltage level of $V_{DD}/2$ (1). To generate random numbers, we first issue one ACT R0 → PRE → ACT R3 command sequence (1). Doing so activates four rows simultaneously and enables charge-sharing between their bitlines. As a result, the bitline ends up with a voltage level outside of reliable sensing margins, e.g., $\sim V_{DD}/2$ (2). The sense amplifier then kicks in and tries to amplify the voltage on the bitline, which results in sampling a random value, e.g., the single depicted bitline is randomly sampled as V_{DD} in this figure (3). Finally, we send a PRE command to complete the process. Note that, to be used as TRNG, rows and bitlines need to be profiled [146, 147].

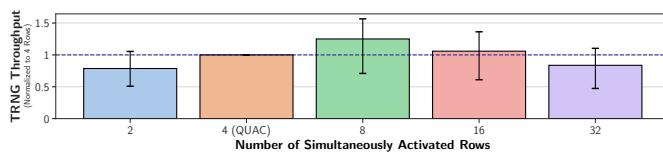


Fig. 13: Throughput of generating true random numbers, as measured in 96 COTS DRAM chips using multiple-row activation, normalized to state-of-the-art DRAM-based TRNG, QUAC-TRNG (i.e., 4-row activation) [146]. Each error bar shows the range across all tested chips. We observe that random numbers that are generated with multiple-row activation and then post-processed with the SHA-256 function [231] pass all NIST STS tests [232], which means 2-, 4-, 8-, 16-, and 32-row activation generates high-quality true random bitstreams. On average, 8- and 16-row activation-based TRNG outperforms the state-of-the-art by $1.25 \times$ and $1.06 \times$, respectively, while 2- and 32-row activation-based TRNG provides $0.69 \times$ and $0.84 \times$ the throughput of the state-of-the-art.

VI. SECTORED DRAM

Two key coarse-grained access mechanisms lead to wasted energy in modern DRAM chips, also impacting PiM efficiency and programmability: large and fixed-size i) data transfers between DRAM and the memory controller and ii) DRAM row activations. Sectored DRAM is designed from the ground up as a low-overhead DRAM substrate (Fig. 14) that reduces wasted energy by enabling fine-grained DRAM data transfer and DRAM row activation. The major idea is to segment the wordlines such that smaller granularity structures are enabled for a given DRAM access. Our results show that Sectored DRAM, compared to a conventional DRAM system, reduces the DRAM energy consumption of data intensive workloads by up to 33% (20% on average), while improving performance by up to 36% (17% on average). Sectored DRAM’s DRAM energy savings combined with its system performance improvement allows system-wide energy savings of up to 23%.

VII. CONCLUSION

We highlighted several recent major advances in Processing-in-DRAM, which demonstrate the promising potential of using and enhancing DRAM as a computation substrate. These works also highlight that DRAM (and in general, memory) should be designed, used, and programmed not as an inactive storage substrate, which is *business as usual* in modern systems, but instead as a *combined computation and storage substrate where both computational capability and storage density are key goals*. Although many challenges remain to enable widespread adoption of Processing-in-DRAM (and Processing-in-Memory in general),² we believe the mindset and infrastructure shift necessary to enable such a combined computation-storage paradigm remains to be the largest challenge. Overcoming this mindset and infrastructure shift can unleash a fundamentally energy-efficient, high-performance, and sustainable way of designing, using, and programming computing systems.

²Multiple prior works [8, 10, 11] overview these challenges.

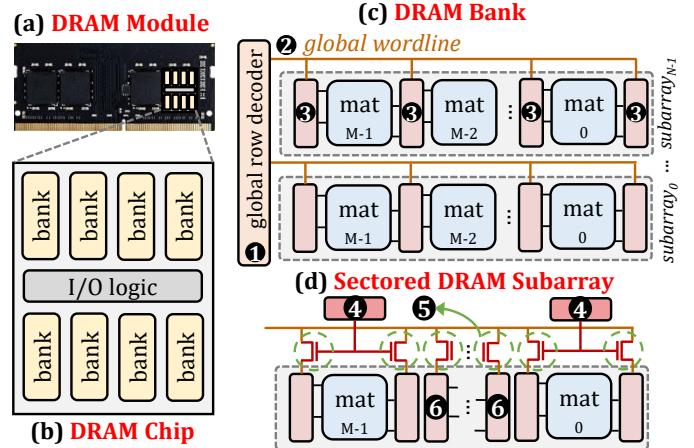


Fig. 14: A DRAM module (a), a DRAM chip with multiple banks (b), baseline DRAM bank organization with multiple subarrays (c), and a Sectored DRAM subarray (d) [222]. The global row decoder (1) enables a global wordline (2) based on the higher-order bits of a DRAM row address (not shown). A global wordline enables a local wordline driver (3) that drives a local wordline using the lower-order bits of the DRAM row address (not shown). The baseline DRAM bank activates all mats in a subarray with one active (ACT) DRAM command. Sectored DRAM implements sector latches (4), sector transistors (5), and additional local wordline drivers (6), allowing the DRAM chip to activate any subset of one or more DRAM mats with an ACT command. To make use of fine-grained DRAM row activation, the memory controller selects sector latches by using the unused bits in the precharge (PRE) command’s encoding [233] to encode the *sector bits*. Each sector bit encodes if a sector latch is set or reset. The memory controller sends a bitvector of sector bits with every PRE command. These sector bits are used for the ACT command that follows the PRE command.

ACKNOWLEDGMENTS

This paper is an extended version of our earlier invited paper and presentation in the “AI Memory” focus session of the IEDM 2024 conference [234]. We thank the anonymous reviewers of IEDM 2024, especially the Memory Technology Committee, for their encouraging feedback. We thank the SAFARI Research Group members for providing a stimulating intellectual and scientific environment. We acknowledge the generous gifts from our industrial partners, including Google, Huawei, Intel, and Microsoft. This work, and our broader work in Processing-in-Memory, is supported in part by the Semiconductor Research Corporation (SRC), the ETH Future Computing Laboratory (EFCL), and the AI Chip Center for Emerging Smart Systems (ACCESS).

REFERENCES

- [1] O. Mutlu, “Memory Scaling: A Systems Architecture Perspective,” in *IMW*, 2013.
- [2] O. Mutlu and L. Subramanian, “Research Problems and Opportunities in Memory Systems,” *SUPERFRI*, 2014.
- [3] J. Dean and L. A. Barroso, “The Tail at Scale,” *CACM*, 2013.
- [4] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, “Profiling a Warehouse-Scale Computer,” in *ISCA*, 2015.
- [5] M. Ferdinand, A. Adileh, O. Kocberber, S. Volos, M. Alisafaei, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, “Clearing the Clouds: A Study of Emerging Scale-Out Workloads on Modern Hardware,” in *ASPLOS*, 2012.
- [6] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, “BigDataBench: A Big Data Benchmark Suite from Internet Services,” in *HPCA*, 2014.
- [7] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, “Enabling Practical Processing in and near Memory for Data-Intensive Computing,” in *DAC*, 2019.
- [8] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, “Processing Data Where It Makes Sense: Enabling In-Memory Computation,” *MicPro*, 2019.
- [9] O. Mutlu, “Intelligent Architectures for Intelligent Machines,” in *VLSI-DAT*, 2020.
- [10] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, “Processing-in-Memory: A Workload-Driven Perspective,” *IBM JRD*, 2019.
- [11] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, “A Modern Primer on Processing in Memory,” in *Emerging Computing: From Devices to Systems — Looking Beyond Moore and Von Neumann*. Springer, 2022.

- [12] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, “DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks,” *IEEE Access*, 2021.
- [13] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Kries, P. Ranganathan *et al.*, “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks,” in *ASPLOS*, 2018.
- [14] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, “Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks,” in *PACT*, 2021.
- [15] S. Wang and E. Ipek, “Reducing Data Movement Energy via Online Data Clustering and Encoding,” in *MICRO*, 2016.
- [16] D. Pandiyar and C.-J. Wu, “Quantifying the Energy Cost of Data Movement for Emerging Smart Phone Workloads on Mobile Platforms,” in *ISWC*, 2014.
- [17] S. Koppula, L. Orosa, A. G. Yağlıkçı, R. Azizi, T. Shahroodi, K. Kanellopoulos, and O. Mutlu, “EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM,” in *MICRO*, 2019.
- [18] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. S. Choi, “Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling,” in *The Memory Forum*, 2014.
- [19] S. A. McKee, “Reflections on the Memory Wall,” in *CF*, 2004.
- [20] M. V. Wilkes, “The Memory Gap and the Future of High Performance Memories,” *CAN*, 2001.
- [21] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, “A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM,” in *ISCA*, 2012.
- [22] W. A. Wulf and S. A. McKee, “Hitting the Memory Wall: Implications of the Obvious,” *CAN*, 1995.
- [23] S. Ghose, T. Li, N. Hajinazar, D. S. Cali, and O. Mutlu, “Demystifying Complex Workload-DRAM Interactions: An Experimental Study,” in *SIGMETRICS*, 2020.
- [24] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing,” in *ISCA*, 2015.
- [25] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, “PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture,” in *ISCA*, 2015.
- [26] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O’Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, “Transparent Offloading and Mapping (TOM) Enabling Programmer-Transparent Near-Data Processing in GPU Systems,” in *ISCA*, 2016.
- [27] Y. Wang, L. Orosa, X. Peng, Y. Guo, S. Ghose, M. Patel, J. S. Kim, J. G. Luna, M. Sadrosadati, N. M. Ghiasi *et al.*, “FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching,” in *MICRO*, 2020.
- [28] R. Sites, “It’s the Memory, Stupid!” *MPR*, 1996.
- [29] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language Models are Few-Shot Learners,” in *NIPS*, 2020.
- [30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *NAACL*, 2019.
- [31] G. F. Oliveira, J. Gómez-Luna, S. Ghose, A. Boroumand, and O. Mutlu, “Accelerating Neural Network Inference with Processing-in-DRAM: From the Edge to the Cloud,” *IEEE Micro*, 2022.
- [32] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, “NEUPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing,” in *ASPLOS*, 2024.
- [33] M. Zhou, W. Xu, J. Kang, and T. Rosing, “TransPIM: A Memory-Based Acceleration via Software-Hardware Co-Design for Transformer,” in *HPCA*, 2022.
- [34] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, “AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference,” in *ASPLOS*, 2024.
- [35] M. Seo, X. T. Nguyen, S. J. Hwang, Y. Kwon, G. Kim, C. Park, I. Kim, J. Park, J. Kim, W. Shin *et al.*, “IANUS: Integrated Accelerator based on NPU-PIM Unified Memory System,” in *ASPLOS*, 2024.
- [36] S. Rhyner, H. Luo, J. Gómez-Luna, M. Sadrosadati, J. Jiang, A. Olgun, H. Gupta, C. Zhang, and O. Mutlu, “PIM-Opt: Demystifying Distributed Optimization Algorithms on a Real-World Processing-In-Memory System,” in *PACT*, 2024.
- [37] S. Yun, K. Kyung, J. Cho, J. Choi, J. Kim, B. Kim, S. Lee, K. Sohn, and J. H. Ahn, “Duplex: A Device for Large Language Models with Mixture of Experts, Grouped Query Attention, and Continuous Batching,” in *MICRO*, 2024.
- [38] H. Jang, J. Song, J. Jung, J. Park, Y. Kim, and J. Lee, “Smart-Infinity: Fast Large Language Model Training using Near-Storage Processing on a Real System,” in *HPCA*, 2024.
- [39] M. Alser, Z. Bingöl, D. S. Cali, J. Kim, S. Ghose, C. Alkan, and O. Mutlu, “Accelerating Genome Analysis: A Primer on an Ongoing Journey,” *IEEE Micro*, 2020.
- [40] G. Singh, M. Alser, D. S. Cali, D. Diamantopoulos, J. Gómez-Luna, H. Corporaal, and O. Mutlu, “FPGA-Based Near-Memory Acceleration of Modern Data-Intensive Applications,” *IEEE Micro*, 2021.
- [41] M. Alser, J. Lindegger, C. Firtina, N. Almadhoun, H. Mao, G. Singh, J. Gomez-Luna, and O. Mutlu, “From Molecules to Genomic Variations: Accelerating Genome Analysis via Intelligent Algorithms and Architectures,” *CSCB*, 2022.
- [42] J. S. Kim, D. S. Cali, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, “GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies,” *BMC Genomics*, 2018.
- [43] N. M. Ghiasi, J. Park, H. Mustafa, J. Kim, A. Olgun, A. Gollwitzer, D. S. Cali, C. Firtina, H. Mao, N. A. Alserr *et al.*, “GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis,” in *ASPLOS*, 2022.
- [44] N. M. Ghiasi, M. Sadrosadati, H. Mustafa, A. Gollwitzer, C. Firtina, J. Eudine, H. Mao, J. Lindegger, M. B. Cavlak, M. Alser *et al.*, “MeGIs: High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing,” in *ISCA*, 2024.
- [45] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand *et al.*, “GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis,” in *MICRO*, 2020.
- [46] D. S. Cali, K. Kanellopoulos, J. Lindegger, Z. Bingöl, G. S. Kalsi, Z. Zuo, C. Firtina, M. B. Cavlak, J. Kim, N. M. Ghiasi *et al.*, “SeGram: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping,” in *ISCA*, 2022.
- [47] D. Senol Cali, J. S. Kim, S. Ghose, C. Alkan, and O. Mutlu, “Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions,” *Briefings in Bioinformatics*, 2018.
- [48] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, “NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules,” in *HPCA*, 2015.
- [49] O. O. Babarinsa and S. Idreos, “JAFAF: Near-Data Processing for Databases,” in *SIGMOD*, 2015.
- [50] F. Devaux, “The True Processing in Memory Accelerator,” in *Hot Chips*, 2019.
- [51] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, “Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-in-Memory Hardware,” in *CUT*, 2021.
- [52] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, “Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System,” *IEEE Access*, 2022.
- [53] C. Giannoula, N. Vijaykumar, N. Papadopoulou, V. Karakostas, I. Fernandez, J. Gómez-Luna, L. Orosa, N. Koziris, G. Goumas, and O. Mutlu, “SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures,” in *HPCA*, 2021.
- [54] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gómez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, “NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling,” in *FPL*, 2020.
- [55] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, N. Kim, Y. Kwon, K. Vladimir, W. Shin, J. Won, M. Lee, H. Joo *et al.*, “A 1ynn 1.25V 8Gb, 16Gb/s/pin GDDR6-Based Accelerator-in-Memory Supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications,” in *ISSCC*, 2022.
- [56] L. Ke, X. Zhang, J. So, J.-G. Lee, S.-H. Kang, S. Lee, S. Han, Y. Cho, J. H. Kim, Y. Kwon *et al.*, “Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM,” *IEEE Micro*, 2021.
- [57] C. Giannoula, I. Fernandez, J. G. Luna, N. Koziris, G. Goumas, and O. Mutlu, “SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-in-Memory Architectures,” in *SIGMETRICS*, 2022.
- [58] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo, “McDRAM: Low Latency and Energy-Efficient Matrix Computations in DRAM,” *TCADICS*, 2018.
- [59] S. Cho, H. Choi, E. Park, H. Shin, and S. Yoo, “McDRAM v2: In-Dynamic Random Access Memory Systolic Array Accelerator to Address the Large Model Problem in Deep Neural Networks on the Edge,” *IEEE Access*, 2020.
- [60] A. Denzler, R. Bera, N. Hajinazar, G. Singh, G. F. Oliveira, J. Gómez-Luna, and O. Mutlu, “Casper: Accelerating Stencil Computation using Near-Cache Processing,” *IEEE Access*, 2023.
- [61] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, “Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems,” in *MICRO*, 2016.
- [62] D. Patterson, T. Anderson, N. Cardwell *et al.*, “A Case for Intelligent RAM,” *IEEE Micro*, 1997.
- [63] D. G. Elliott, M. Stumm, W. M. Snelgrove *et al.*, “Computational RAM: Implementing Processors in Memory,” *D&T*, 1999.

- [64] M. A. Z. Alves, P. C. Santos, F. B. Moreira, and ophthers, "Saving Memory Movements Through Vector Processing in the DRAM," in *CASES*, 2015.
- [65] S. L. Xi, O. Babarinsa, M. Athanassoulis, and S. Idreos, "Beyond the Wall: Near-Data Processing for Databases," in *DaMoN*, 2015.
- [66] W. Sun, Z. Li, S. Yin, S. Wei, and L. Liu, "ABC-DIMM: Alleviating the Bottleneck of Communication in DIMM-Based Near-Memory Processing with Inter-DIMM Broadcast," in *ISCA*, 2021.
- [67] K. K. Matam, G. Koo, H. Zha, H.-W. Tseng, and M. Annavaram, "GraphSSD: Graph Semantics Aware SSD," in *ISCA*, 2019.
- [68] M. Gokhale, B. Holmes, and K. Iobst, "Processing in Memory: The Terasys Massively Parallel PIM Array," *Computer*, 1995.
- [69] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, et al., "Mapping Irregular Applications to DIVA, a PIM-Based Data-Intensive Architecture," in *SC*, 1999.
- [70] M. A. Z. Alves, P. C. Santos, M. Diener, and L. Carro, "Opportunities and Challenges of Performing Vector Operations Inside the DRAM," in *MEMSYS*, 2015.
- [71] E. Lockerman, A. Feldmann, M. Bakhsalipour, A. Stanescu, S. Gupta, D. Sanchez, and N. Beckmann, "Livia: Data-Centric Computing Throughout the Memory Hierarchy," in *ASPLOS*, 2020.
- [72] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," in *HPCA*, 2017.
- [73] A. Boroumand, S. Ghose, B. Lucia, K. Hsieh, K. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *CAL*, 2017.
- [74] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-Oriented Programmable Processing in Memory," in *IPDC*, 2014.
- [75] M. Gao and C. Kozyrakis, "HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing," in *HPCA*, 2016.
- [76] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel, B. Falsafi, B. Grot, and D. Pnevmatikatos, "The Mondrian Data Engine," in *ISCA*, 2017.
- [77] P. C. Santos, G. F. Oliveira, D. G. Tomé, M. A. Z. Alves, E. C. Almeida, and L. Carro, "Operand Size Reconfiguration for Big Data Processing in Memory," in *DATE*, 2017.
- [78] G. F. Oliveira, P. C. Santos, M. A. Alves, and L. Carro, "NIM: An HMC-Based Machine for Neuron Computation," in *ARC*, 2017.
- [79] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ASPLOS*, 2017.
- [80] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *ISCA*, 2016.
- [81] P. Gu, S. Li, D. Stow, R. Barnes, L. Liu, Y. Xie, and E. Kursun, "Leveraging 3D Technologies for Hardware Security: Opportunities and Challenges," in *GLSVLSI*, 2016.
- [82] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, R. Ausavarungnirun, K. Hsieh, N. Hajinazar, K. T. Malladi, H. Zheng, et al., "CoNDÁ: Efficient Cache Coherence Support for Near-Data Accelerators," in *ISCA*, 2019.
- [83] S. H. Pugsley, J. Jesters, H. Zhang, R. Balasubramonian, et al., "NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads," in *ISPASS*, 2014.
- [84] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities," in *PACT*, 2016.
- [85] B. Akin, F. Franchetti, and J. C. Hoe, "Data Reorganization in Memory Using 3D-Stacked DRAM," in *ISCA*, 2015.
- [86] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.
- [87] J. H. Lee, J. Sim, and H. Kim, "BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models," in *PACT*, 2015.
- [88] A. Boroumand, S. Ghose, G. F. Oliveira, and O. Mutlu, "Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design," in *ICDE*, 2022.
- [89] A. Boroumand, "Practical Mechanisms for Reducing Processor-Memory Data Movement in Modern Workloads," Ph.D. dissertation, Carnegie Mellon University, 2020.
- [90] M. Besta, R. Kanakagiri, G. Kwasniewski, R. Ausavarungnirun, J. Beránek, K. Kanellopoulos, K. Janda, Z. Vonarbburg-Shmaria, L. Gianinazzi, I. Stefan, et al., "SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems," in *MICRO*, 2021.
- [91] I. Fernandez, R. Quislant, E. Gutiérrez, O. Plata, C. Giannoula, M. Alser, J. Gómez-Luna, and O. Mutlu, "NATSA: A Near-Data Processing Accelerator for Time Series Analysis," in *ICCD*, 2020.
- [92] G. Singh, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning," in *DAC*, 2019.
- [93] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim, et al., "A 20nm 6GB Function-in-Memory DRAM, Based on HBM2 with a 1.2 TFLOPS Programmable Computing Unit using Bank-Level Parallelism, for Machine Learning Applications," in *ISSCC*, 2021.
- [94] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, et al., "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product," in *ISCA*, 2021.
- [95] D. Niu, S. Li, Y. Wang, W. Han, Z. Zhang, Y. Guan, T. Guan, F. Sun, F. Xue, L. Duan, et al., "184QPS/W 64Mb/mm² 3D Logic-to-DRAM Hybrid Bonding with Process-Near-Memory Engine for Recommendation System," in *ISSCC*, 2022.
- [96] Q. Zhu, T. Graf, H. E. Sumbul, L. Pileggi, and F. Franchetti, "Accelerating Sparse Matrix-Matrix Multiplication with 3D-Stacked Logic-in-Memory Hardware," in *HPEC*, 2013.
- [97] E. Azarkhish, C. Pfister, D. Rossi, I. Loi, and L. Benini, "Logic-Based Interconnect Design for Near Memory Computing in the Smart Memory Cube," *IEEE VLSI*, 2016.
- [98] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and Energy Efficient Deep Learning with Smart Memory Cubes," *TPDS*, 2018.
- [99] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T. M. Low, L. Pileggi, J. C. Hoe, and F. Franchetti, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WondP*, 2014.
- [100] B. Akin, J. C. Hoe, and F. Franchetti, "HAMLeT: Hardware Accelerated Memory Layout Transform within 3D-Stacked DRAM," in *HPEC*, 2014.
- [101] Y. Huang, L. Zheng, P. Yao, J. Zhao, X. Liao, H. Jin, and J. Xue, "A Heterogeneous PIM Hardware-Software Co-Design for Energy-Efficient Graph Processing," in *IPDPS*, 2020.
- [102] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "GraphH: A Processing-in-Memory Architecture for Large-Scale Graph Processing," *TCAD*, 2018.
- [103] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-Memory for Energy-Efficient Neural Network Training: A Heterogeneous Approach," in *MICRO*, 2018.
- [104] P.-A. Tsai, C. Chen, and D. Sanchez, "Adaptive Scheduling for Systems with Asymmetric Memory Hierarchies," in *MICRO*, 2018.
- [105] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, "iPIM: Programmable In-Memory Image Processing Accelerator using Near-Bank Architecture," in *ISCA*, 2020.
- [106] A. Farmahini-Farahani, J. H. Ahn, K. Compton, and N. S. Kim, "DRAMA: An Architecture for Accelerated Processing Near Memory," *CAL*, 2014.
- [107] H. Asghari-Moghaddam, A. Farmahini-Farahani, K. Morrow, et al., "Near-DRAM Acceleration with Single-ISA Heterogeneous Processing in Standard Memory Modules," *IEEE Micro*, 2016.
- [108] J. Huang, R. R. Puli, P. Majumder, S. Kim, R. Boyapati, K. H. Yum, and E. J. Kim, "Active-Routing: Compute on the Way for Near-Data Processing," in *HPCA*, 2019.
- [109] C. D. Kersey, H. Kim, and S. Yalamanchili, "Lightweight SIMD Core Designs for Intelligent 3D Stacked DRAM," in *MEMSYS*, 2017.
- [110] J. Li, X. Wang, A. Tumeo, B. Williams, J. D. Leidel, and Y. Chen, "PIMS: A Lightweight Processing-in-Memory Accelerator for Stencil Computations," in *MEMSYS*, 2019.
- [111] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian, "GraphQ: Scalable PIM-Based Graph Processing," in *MICRO*, 2019.
- [112] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition," in *HPCA*, 2018.
- [113] H. Lim and G. Park, "Triple Engine Processor (TEP): A Heterogeneous Near-Memory Processor for Diverse Kernel Operations," *TACO*, 2017.
- [114] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "A Case for Near Memory Computation Inside the Smart Memory Cube," in *EMS*, 2016.
- [115] M. A. Z. Alves, M. Diener, P. C. Santos, and L. Carro, "Large Vector Extensions Inside the HMC," in *DATE*, 2016.
- [116] J. Jang, J. Heo, Y. Lee, J. Won, S. Kim, S. J. Jung, H. Jang, T. J. Ham, and J. W. Lee, "Charon: Specialized Near-Memory Processing Architecture for Clearing Dead Objects in Memory," in *MICRO*, 2019.
- [117] R. Nair, S. F. Antao, C. Bertolli, P. Bose, et al., "Active Memory Cube: A Processing-in-Memory Architecture for Exascale Systems," *IBM JRD*, 2015.
- [118] R. Hadidi, L. Nai, H. Kim, and H. Kim, "CAIRO: A Compiler-Assisted Technique for Enabling Instruction-Level Offloading of Processing-in-Memory," *TACO*, 2017.
- [119] P. C. Santos, G. F. Oliveira, J. P. Lima, M. A. Alves, L. Carro, and A. C. Beck, "Processing in 3D Memories to Speed Up Operations on Complex Data Structures," in *DATE*, 2018.
- [120] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *ISCA*, 2016.
- [121] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Sri Kumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.

- [122] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
- [123] V. Seshadri and O. Mutlu, "In-DRAM Bulk Bitwise Execution Engine," arXiv:1905.09822, 2019.
- [124] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator," in *MICRO*, 2017.
- [125] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
- [126] V. Seshadri and O. Mutlu, "The Processing Using Memory Paradigm: In-DRAM Bulk Copy, Initialization, Bitwise AND and OR," arXiv:1610.09603, 2016.
- [127] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "DrAcc: A DRAM Based Accelerator for Accurate CNN Inference," in *DAC*, 2018.
- [128] X. Xin, Y. Zhang, and J. Yang, "ELP2IM: Efficient and Low Power Bitwise Operation Processing in DRAM," in *HPCA*, 2020.
- [129] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating Graph Processing Using ReRAM," in *HPCA*, 2018.
- [130] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *HPCA*, 2017.
- [131] F. Gao, G. Tziantzoulis, and D. Wentzlaff, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs," in *MICRO*, 2019.
- [132] C. Eckert, X. Wang, J. Wang, A. Subramanyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks," in *ISCA*, 2018.
- [133] S. Aga, S. Jeloka, A. Subramanyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute Caches," in *HPCA*, 2017.
- [134] D. Fujiki, S. Mahlke, and R. Das, "Duality Cache for Data Parallel Acceleration," in *ISCA*, 2019.
- [135] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," arXiv:1611.09988, 2016.
- [136] V. Seshadri and O. Mutlu, "Simple Operations in Memory to Reduce Data Movement," in *Advances in Computers, Volume 106*, 2017.
- [137] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch *et al.*, "RowClone: Accelerating Data Movement and Initialization Using DRAM," arXiv:1805.03502, 2018.
- [138] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.
- [139] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories," in *DAC*, 2016.
- [140] J. D. Ferreira, G. Falcao, J. Gómez-Luna, M. Alser, L. Orosa, M. Sadrosadati, J. S. Kim, G. F. Oliveira, T. Shahroodi, A. Nori *et al.*, "pLUTO: Enabling Massively Parallel Computation in DRAM via Lookup Tables," in *MICRO*, 2022.
- [141] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision," in *ISCA*, 2019.
- [142] Z. He, L. Yang, S. Angizi, A. S. Rakin, and D. Fan, "Sparse BD-Net: A Multiplication-Less DNN with Sparse Binarized Depth-Wise Separable Convolution," *JETC*, 2020.
- [143] J. Park, R. Azizi, G. F. Oliveira, M. Sadrosadati, R. Nadig, D. Novo, J. Gómez-Luna, M. Kim, and O. Mutlu, "Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory," in *MICRO*, 2022.
- [144] M. S. Truong, L. Shen, A. Glass, A. Hoffmann, L. R. Carley, J. A. Bain, and S. Ghose, "Adapting the RACER Architecture to Integrate Improved In-ReRAM Logic Primitives," *JETCAS*, 2022.
- [145] M. S. Truong, E. Chen, D. Su, L. Shen, A. Glass, L. R. Carley, J. A. Bain, and S. Ghose, "RACER: Bit-Pipelined Processing Using Resistive Memory," in *MICRO*, 2021.
- [146] A. Olgun, M. Patel, A. G. Yağlıkçı, H. Luo, J. S. Kim, F. N. Bostancı, N. Vijaykumar, O. Ergin, and O. Mutlu, "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAMs," in *ISCA*, 2021.
- [147] J. S. Kim, M. Patel, H. Hassan, L. Orosa, and O. Mutlu, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers With Low Latency and High Throughput," in *HPCA*, 2019.
- [148] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices," in *HPCA*, 2018.
- [149] F. N. Bostancı, A. Olgun, L. Orosa, A. G. Yağlıkçı, J. S. Kim, H. Hassan, O. Ergin, and O. Mutlu, "DR-STRaNGe: End-to-End System Design for DRAM-Based True Random Number Generators," in *HPCA*, 2022.
- [150] A. Olgun, J. G. Luna, K. Kanellopoulos, B. Salami, H. Hassan, O. Ergin, and O. Mutlu, "PiDRAM: A Holistic End-to-End FPGA-Based Framework for Processing-in-DRAM," *TACO*, 2022.
- [151] M. F. Ali, A. Jaiswal, and K. Roy, "In-Memory Low-Cost Bit-Serial Addition Using Commodity DRAM Technology," in *TCAS-I*, 2019.
- [152] S. Angizi and D. Fan, "GraphiDe: A Graph Processing Accelerator Leveraging In-DRAM-Computing," in *GLSVLSI*, 2019.
- [153] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator," in *MICRO*, 2018.
- [154] A. Subramanyan and R. Das, "Parallel Automata Processor," in *ISCA*, 2017.
- [155] Y. Zha and J. Li, "Hyper-AP: Enhancing Associative Processing Through A Full-Stack Optimization," in *ISCA*, 2020.
- [156] D. Fujiki, S. Mahlke, and R. Das, "In-Memory Data Parallel Processor," in *ASPLOS*, 2018.
- [157] L. Orosa, Y. Wang, M. Sadrosadati, J. Kim, M. Patel, I. Puddu, H. Luo, K. Razavi, J. Gómez-Luna, H. Hassan, N. M. Ghiasi, S. Ghose, and O. Mutlu, "CODIC: A Low-Cost Substrate for Enabling Custom In-DRAM Functionalities and Optimizations," in *ISCA*, 2021.
- [158] M. Sharad, D. Fan, and K. Roy, "Ultra Low Power Associative Computing with Spin Neurons and Resistive Crossbar Memory," in *DAC*, 2013.
- [159] S. H. S. Rezaei, M. Modarressi, R. Ausavarungnirun, M. Sadrosadati, O. Mutlu, and M. Daneshtalab, "NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories," *CAL*, 2020.
- [160] C. Gao, X. Xin, Y. Lu, Y. Zhang, J. Yang, and J. Shu, "ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory Based SSDs," in *MICRO*, 2021.
- [161] W. H. Choi, P.-F. Chiu, W. Ma, G. Hemink, T. T. Hoang, M. Lueker-Boden, and Z. Bandic, "An In-Flash Binary Neural Network Accelerator with SLC NAND Flash Array," in *ISCA*, 2020.
- [162] R. Han, P. Huang, Y. Xiang, C. Liu, Z. Dong, Z. Su, Y. Liu, L. Liu, X. Liu, and J. Kang, "A Novel Convolution Computing Paradigm Based on NOR Flash Array with High Computing Speed and Energy Efficiency," *TCAS-I*, 2019.
- [163] F. Merrikh-Bayat, X. Guo, M. Klachko, M. Prezioso, K. K. Likharev, and D. B. Strukov, "High-Performance Mixed-Signal Neurocomputing with Nanoscale Floating-Gate Memory Cell Arrays," *TNNLS*, 2017.
- [164] P. Wang, F. Xu, B. Wang, B. Gao, H. Wu, H. Qian, and S. Yu, "Three-Dimensional NAND Flash for Vector-Matrix Multiplication," *TVLSI*, 2018.
- [165] H.-T. Lue, P.-K. Hsu, M.-L. Wei, T.-H. Yeh, P.-Y. Du, W.-C. Chen, K.-C. Wang, and C.-Y. Lu, "Optimal Design Methods to Transform 3D NAND Flash into a High-Density, High-Bandwidth and Low-Power Nonvolatile Computing in Memory (nvCIM) Accelerator for Deep-Learning Neural Networks (DNN)," in *IEDM*, 2019.
- [166] S. Kim, Y. Jin, G. Sohn, J. Bae, T. J. Ham, and J. W. Lee, "Behemoth: A Flash-Centric Training Accelerator for Extreme-Scale DNNs," in *FAST*, 2021.
- [167] S. Wang, "MemCore: Computing-in-Flash Design for Deep Neural Network Acceleration," in *EDTM*, 2022.
- [168] R. Han, Y. Xiang, P. Huang, Y. Shan, X. Liu, and J. Kang, "Flash Memory Array for Efficient Implementation of Deep Neural Networks," *Adv. Intell. Syst.*, 2021.
- [169] M. Kang, H. Kim, H. Shin, J. Sim, K. Kim, and L.-S. Kim, "S-FLASH: A NAND Flash-Based Deep Neural Network Accelerator Exploiting Bit-Level Sparsity," *TC*, 2021.
- [170] S.-T. Lee and J.-H. Lee, "Neuromorphic Computing Using NAND Flash Memory Architecture with Pulse Width Modulation Scheme," *Front. Neurosci.*, 2020.
- [171] H. Lee, M. Kim, D. Min, J. Kim, J. Back, H. Yoo, J.-H. Lee, and J. Kim, "3D-DFIM: An Extreme Energy-Efficient DNN Acceleration System Using 3D NAND Flash-Based In-Situ PIM Unit," in *MICRO*, 2022.
- [172] X. Si, W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Sun, R. Liu, S. Yu, H. Yamauchi, Q. Li, and M.-F. Chang, "A Dual-Split 6T SRAM-Based Computing-in-Memory Unit-Macro with Fully Parallel Product-Sum Operation for Binarized DNN Edge Processors," *TCAS-I*, 2019.
- [173] W. A. Simon, Y. M. Qureshi, M. Rios, A. Levisse, M. Zapater, and D. Atienza, "BLADE: An In-Cache Computing Architecture for Edge Devices," *TC*, 2020.
- [174] A. Nag, C. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardson, "GenCache: Leveraging In-Cache Operators for Efficient Sequence Alignment," in *MICRO*, 2019.
- [175] X. Wang, J. Yu, C. Augustine, R. Iyer, and R. Das, "Bit Prudent In-Cache Acceleration of Deep Convolutional Neural Networks," in *HPCA*, 2019.
- [176] K. Al-Hawaj, O. Afuye, S. Agwa, A. Apsel, and C. Batten, "Towards a Reconfigurable Bit-Serial/Bit-Parallel Vector Accelerator Using In-Situ Processing-in-SRAM," in *ISCAS*, 2020.
- [177] M. Kang, M.-S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, "An Energy-Efficient VLSI Architecture for Pattern Recognition via Deep Embedding of Computation in SRAM," in *ICASSP*, 2014.
- [178] H. Kim, T. Yoo, T. T.-H. Kim, and B. Kim, "Colonnade: A Reconfigurable SRAM-Based Digital Bit-Serial Compute-in-Memory Macro for Processing Neural Networks," *JSSC*, 2021.
- [179] Z. Jiang, S. Yin, J.-S. Seo, and M. Seok, "C3SRAM: An In-Memory-Computing SRAM Macro Based on Robust Capacitive Coupling Computing Mechanism," *JSSC*, 2020.

- [180] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, “A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-in-Memory,” *JSSC*, 2016.
- [181] Z. Wang, C. Liu, A. Arora, L. John, and T. Nowatzki, “Infinity Stream: Portable and Programmer-Friendly In-/Near-Memory Fusion,” in *ASPLOS*, 2023.
- [182] M. Kang, E. P. Kim, M.-s. Keel, and N. R. Shanbhag, “Energy-Efficient and High Throughput Sparse Distributed Memory Architecture,” in *ISCAS*, 2015.
- [183] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, “DUAL: Acceleration of Clustering Algorithms Using Digital-Based Processing in-Memory,” in *MICRO*, 2020.
- [184] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, “Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM,” in *HPCA*, 2016.
- [185] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, “SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM,” in *ASPLOS*, 2021.
- [186] Q. Deng, Y. Zhang, M. Zhang, and J. Yang, “LAcc: Exploiting Lookup Table-Based Fast and Accurate Vector Multiplication in DRAM-Based CNN Accelerator,” in *DAC*, 2019.
- [187] P. R. Sutradhar, S. Bavikadi, M. Connolly, S. Prajapati, M. A. Indovina, S. M. P. Dinakarao, and A. Ganguly, “Look-Up-Table Based Processing-in-Memory Architecture with Programmable Precision-Scaling for Deep Learning Applications,” *TPDS*, 2021.
- [188] P. R. Sutradhar, M. Connolly, S. Bavikadi, S. M. P. Dinakarao, M. A. Indovina, and A. Ganguly, “pPIM: A Programmable Processor-in-Memory Architecture with Precision-Scaling for Deep Learning,” *CAL*, 2020.
- [189] M. Lenjani, P. Gonzalez, E. Sadredini, S. Li, Y. Xie, A. Akel, S. Eilert, M. R. Stan, and K. Skadron, “Fulcrum: A Simplified Control and Access Mechanism Toward Flexible and Practical In-Situ Accelerators,” in *HPCA*, 2020.
- [190] X. Peng, Y. Wang, and M.-C. Yang, “CHOPPER: A Compiler Infrastructure for Programmable Bit-Serial SIMD Processing Using Memory In DRAM,” in *HPCA*, 2023.
- [191] G. F. Oliveira, A. Kohli, D. Novo, J. Gómez-Luna, and O. Mutlu, “DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures,” arXiv:2310.10168, 2023.
- [192] G. F. Oliveira, J. Gómez-Luna, S. Ghose, and O. Mutlu, “Methodologies, Workloads, and Tools for Processing-in-Memory: Enabling the Adoption of Data-Centric Architectures,” in *ISVLSI*, 2022.
- [193] G. F. Oliveira, A. Boroumand, S. Ghose, J. Gómez-Luna, and O. Mutlu, “Heterogeneous Data-Centric Architectures for Modern Data-Intensive Applications: Case Studies in Machine Learning and Databases,” in *ISVLSI*, 2022.
- [194] T. Shahroodi, G. Singh, M. Zahedi, H. Mao, J. Lindegger, C. Firtina, S. Wong, O. Mutlu, and S. Hamdioui, “Swordfish: A Framework for Evaluating Deep Neural Network-Based Basecalling Using Computation-In-Memory with Non-Ideal Memristors,” in *MICRO*, 2023.
- [195] J. Chen, J. Gómez-Luna, I. E. Hajj, Y. Guo, and O. Mutlu, “SimplePIM: A Software Framework for Productive and Efficient Processing-In-Memory,” in *PACT*, 2023.
- [196] H. Gupta, M. Kabra, J. Gómez-Luna, K. Kanellopoulos, and O. Mutlu, “Evaluating Homomorphic Operations on a Real-World Processing-In-Memory System,” in *IISWC*, 2023.
- [197] J. Gómez-Luna, Y. Guo, S. Brocard, J. Legriel, R. Cimadomo, G. F. Oliveira, G. Singh, and O. Mutlu, “Evaluating Machine Learning Workloads on Memory-Centric Computing Systems,” in *ISPASS*, 2023.
- [198] M. Item, J. Gómez-Luna, Y. Guo, G. F. Oliveira, M. Sadrosadati, and O. Mutlu, “TransPimLib: Efficient Transcendental Functions for Processing-in-Memory Systems,” in *ISPASS*, 2023.
- [199] S. Diab, A. Nassereldine, M. Alser, J. Gómez Luna, O. Mutlu, and I. El Hajj, “A Framework for High-Throughput Sequence Alignment Using Real Processing-In-Memory Systems,” *Bioinformatics*, 2023.
- [200] H. Mao, M. Alser, M. Sadrosadati, C. Firtina, A. Baranwal, D. S. Cali, A. Manglik, N. A. Alserr, and O. Mutlu, “GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping,” in *MICRO*, 2022.
- [201] G. Singh, D. Diamantopoulos, J. Gómez-Luna, C. Hagleitner, S. Stuijk, H. Corporaal, and O. Mutlu, “Accelerating Weather Prediction Using Near-Memory Reconfigurable Fabric,” *TRETS*, 2022.
- [202] W. H. Kautz, “Cellular logic-in-memory arrays,” *IEEE ToC*, 1969.
- [203] H. S. Stone, “A Logic-in-Memory Computer,” *IEEE ToC*, 1970.
- [204] HMC Consortium, “HMC Specification Rev. 2.0,” www.hybridmemorycube.org/.
- [205] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin *et al.*, “A 1.2V 8Gb 8-Channel 128GB/s High-Bandwidth Memory (HBM) Stacked DRAM with Effective Microbump I/O Test Methods Using 29nm Process and TSV,” in *ISSCC*, 2014.
- [206] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, “Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost,” *TACO*, 2016.
- [207] JEDEC, *JESD23-5D: High Bandwidth Memory (HBM) DRAM Standard*, 2021.
- [208] JEDEC, *JESD23-8A: High Bandwidth Memory (HBM3) DRAM Standard*, 2021.
- [209] K. Kim and M.-j. Park, “Present and Future, Challenges of High Bandwidth Memory (HBM),” in *IMW*, 2024.
- [210] I. Fernandez, C. Giannoula, A. Manglik, R. Quislant, N. M. Ghiasi, J. Gómez-Luna, E. Gutierrez, O. Plata, and O. Mutlu, “MATSA: An MRAM-Based Energy-Efficient Accelerator for Time Series Analysis,” *IEEE Access*, 2024.
- [211] J. P. C. de Lima, P. C. Santos, M. A. Alves, A. Beck, and L. Carro, “Design Space Exploration for PIM Architectures in 3D-Stacked Memories,” in *CF*, 2018.
- [212] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, “BlueDBM: An appliance for big data analytics,” *ISCA*, 2015.
- [213] R. H. Dennard, “Field-Effect Transistor Memory,” U.S. Patent 3,387,286, 1968.
- [214] K. Gogineni, S. S. Dayapule, J. Gómez-Luna, K. Gogineni, P. Wei, T. Lan, M. Sadrosadati, O. Mutlu, and G. Venkataramani, “SwiftRL: Towards Efficient Reinforcement Learning on Real Processing-In-Memory Systems,” in *ISPASS*, 2024.
- [215] W. Xiong, L. Ke, D. Jankov, M. Kounavis, X. Wang, E. Northup, J. A. Yang, B. Acun, C.-J. Wu, P. T. P. Tang *et al.*, “SecNDP: Secure Near-Data Processing with Untrusted Memory,” in *HPCA*, 2022.
- [216] S. Aga and S. Narayanasamy, “Invisimem: Smart Memory Defenses For Memory Bus Side Channel,” *ISCA*, 2017.
- [217] F. Gao, G. Tziantzioulis, and D. Wentzlaff, “FracDRAM: Fractional Values in Off-the-Shelf DRAM,” in *MICRO*, 2022.
- [218] A. Olgun, H. Hassan, A. G. Yağlıkçı, Y. C. Tuğrul, L. Orosa, H. Luo, M. Patel, O. Ergin, and O. Mutlu, “DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips,” *TCAD*, 2023.
- [219] G. F. Oliveira, A. Olgun, A. G. G. Yağlıkçı, N. Bostancı, J. Gómez-Luna, S. Ghose, and O. Mutlu, “MIMDRAM: An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Computing,” in *HPCA*, 2024.
- [220] I. E. Yuksel, Y. C. Tugrul, A. Olgun, F. N. Bostancı, A. G. Yağlıkçı, G. F. de Oliveira, H. Luo, J. G. Luna, M. Sadrosadati, and O. Mutlu, “Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis,” in *HPCA*, 2024.
- [221] I. E. Yuksel, Y. C. Tugrul, F. N. Bostancı, G. F. de Oliveira, A. G. Yağlıkçı, A. Olgun, M. Soysal, H. Luo, J. G. Luna, M. Sadrosadati, and O. Mutlu, “Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips: Experimental Characterization and Analysis,” in *DSN*, 2024.
- [222] A. Olgun, F. Bostancı, G. F. Oliveira, Y. C. Tugrul, R. Bera, A. G. Yağlıkçı, H. Hassan, O. Ergin, and O. Mutlu, “Sectedor DRAM: An Energy-Efficient High-Throughput and Practical Fine-Grained DRAM Architecture,” *TACO*, 2024.
- [223] I. E. Yuksel, Y. C. Tugrul, F. N. Bostancı, A. G. Yağlıkçı, A. Olgun, G. F. Oliveira, M. Soysal, H. Luo, J. G. Luna, M. Sadrosadati, and O. Mutlu, “PULSAR: Simultaneous Many-Row Activation for Reliable and High-Performance Computing in Off-the-Shelf DRAM Chips,” arXiv:2312.02880, 2023.
- [224] M. J. Flynn, “Very High-Speed Computing Systems,” *Proc. IEEE*, 1966.
- [225] Intel Corp., “6th Generation Intel Core Processor Family Datasheet,” <http://www.intel.com/content/www/us/en/processors/core/>.
- [226] NVIDIA, “NVIDIA A100 Tensor Core GPU Architecture,” <https://t.ly/fMUgA>, 2020.
- [227] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, “SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies,” in *HPCA*, 2017.
- [228] F. Bai, S. Wang, X. Jia, Y. Guo, B. Yu, H. Wang, C. Lai, Q. Ren, and H. Sun, “A Low-Cost Reduced-Latency DRAM Architecture with Dynamic Reconfiguration of Row Decoder,” *TVLSI*, 2022.
- [229] N. H. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson Education India, 2015.
- [230] M. A. Turi and J. G. Delgado-Frias, “High-Performance Low-Power Selective Precharge Schemes for Address Decoders,” *TCAS-II*, 2008.
- [231] National Institute of Standards and Technology (NIST), “180-2: Secure Hash Standard (SHS),” *Federal Information Processing Standards Publications*, 2012.
- [232] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, S. Leigh, M. Levenson, M. Vangel, N. Heckert, and D. Banks, “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications,” *NIST SP*, 2010.
- [233] JEDEC, *JESD79-4C: DDR4 SDRAM*, 2017.
- [234] O. Mutlu, G. F. Oliveira, A. Olgun, and I. E. Yuksel, “Memory-Centric Computing: Recent Advances in Processing-in-DRAM (Invited),” in *IEDM*, 2024.